MDA Applied: From Sequence Diagrams to Web Service Choreography

Bernhard Bauer[1] and Jörg P. Müller[2]

[1] Programming of Distributed Systems, Institute of Computer Science,
University of Augsburg, D-86135 Augsburg, Germany
bernhard.bauer@informatik.uni-augsburg.de

[2] Siemens AG Corporate Technology, Intelligent Autonomous Systems,
Otto-Hahn-Ring 6, D-81739 München, Germany
joerg.p.mueller@siemens.com

Abstract. Web Services and Web Service composition languages for Web Service choreography are becoming more and more important in the area for inter-enterprise application and process integration. However the aspects of modeling these software systems have not been studied in detail, in contrast to the definition of business processes where well-known techniques exist. The model-driven architecture (MDA) approach of the Object Management Group is a good starting point for the development of Web Services and Web Service choreography. In this paper we show how platform independent models specified by UML 2 sequence diagrams can be automatically transformed in a Web Service composition language representation.

1 Introduction

Over the past few years, enterprises are currently in a thorough transformation process as they encounter the necessity to react to challenges such as globalization, unstable varying demand, and mass customization. A most important factor to maintaining competitiveness is the ability of an enterprise to describe, standardize, and adapt the way it reacts to certain types of business events, and how it interacts as well as its procedures for interaction with suppliers, partners, competitors, and customers. Today, virtually all larger enterprises describe these procedures and interactions in terms of *business processes,* and invest huge efforts to describe and standardize these processes. Web Services are the key technology for Enterprise Application Integration (i.e. EAI; see e.g. [3] for details, [16]) and Inter Enterprise Integration. IBM defines Web Services as [1]: *"Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web* (see e.g. [17])." It is possible to combine them to added-value Web Services offering more functionality than the original ones. This process is called Web Service Choreography or -composition depending on the point of view of the description. Several standards are under development for the definition of languages for Web Service composition or Web Service Choreography, typical examples are BPEL4WS [4], BPML [9], XPDL [13]. ebXML [8] with ebXML *Business Process Specification Schema* [7].

The **Model Driven Architecture (MDA)** (for details see [10, 11]) is a framework for software development driven by the OMG. The following models are at the core of the MDA: **Computational Independent Model (CIM):** This model describes the business logic and domain model; **Platform Independent Model (PIM):** This model is defined at a high level of abstraction; it is independent of any implementation technology. **Platform Specific Model (PSM):** It is tailored to specify a system in terms of the implementation constructs available in one specific implementation technology, e.g. Web Services. **Code**: The final step in the development is the transformation of each PSM to code. Based on OMG's model-driven approach, our objective is to demonstrate a mapping of platform independent models based on UML 2 sequence diagrams [12] (triggered by e.g. [15]) to a platform dependent model based on the Business Process Execution Language for Web Services (BPEL4WS). This paper can be seen as part of a series of papers dealing with software engineering starting from business processes and transforming them into web services choreography, see [19, 18].

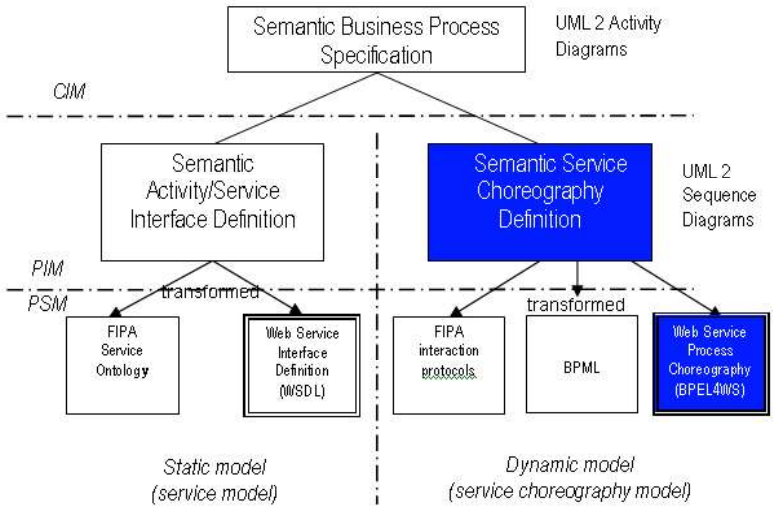


Fig. 1. Web Service enabled business processes

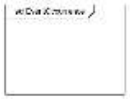
2 Conceptual Methodology

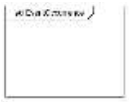
Figure 1 (an updated version of Figure 3 of [18]) illustrates the top-down development process starting with a semantic business process specification using and extending UML 2.0 activity diagrams. This specification is refined into two models: a **static model**, which is essentially the service model in our conceptual methodology, even though enhanced with metadata, such as the description of pre- and post-conditions for service invocation, and with exception definitions; a **dynamic model**, which is essentially the service choreography oriented layer in the conceptual methodology. Each of these two models is described by a platform independent model and one or more platform specific models. In this paper we will focus on the mapping of UML 2 Sequence Diagrams to BPEL4WS as a part of the development process. Readers interested in the other parts are referred to [18, 19].

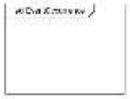
3 From Sequence Diagrams to BPEL4WS

As a next step we will now go into the details of UML 2 sequence diagrams and define informally[1] by induction the mapping of sequence diagram elements to BPEL2WS, i.e. a mapping

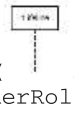
transform: Sequence Diagram Element → BPEL4WS

A sequence diagram defines an interaction denoted as  . Thus a complete sequence diagram is transformed into a process definition of BPEL4WS:

transform () = <process name = "EventOccurence" >

transform(inner_part())</process>

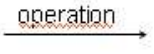
where `inner_part` delivers the sub-diagram defined in the overall sequence diagram.
Lifelines are a modeling element that represents an individual participant in an interaction. A lifeline represents only one interacting entity. They are transformed by the following rule:



transform (…) = <partners> partner name = "Lifeline" serviceLinkType = "…"
partnerRole = "…" myRole = "…" </partner> … </partners>

Note, that the `serviceLinkType`, `partnerRole` as well as `myRole` are not specified in the sequence diagram, but have to be defined in a e.g. class diagram defining the role of a participant and the interface (`serviceLinkType`) as well as the partner role.
Messages are translated as follows
Synchronous/Asynchronous messages:

Transform(

) = <receive partner = receiver(

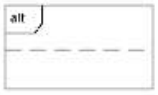
)
portType = "…" operation = "operation" inputContainer = "operationInC"
outputContainer = "operationOutC" </receive>

where `receiver` calculates the name of the right-hand-side lifeline name and `operationInC` and `operationOutC` are automatically generated tokens for the input and output container of the operation, for aynchronous messages an output container is not specified since no result is transported back to the sender.
Reply messages:

Transform( ) = <reply partner = receiver( ) portType = "…"
operation = "operation" container = "operationC"</reply>

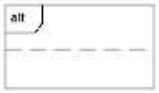
where `receiver` calculates the name of the left-hand-side lifeline name and `operationC` is an automatically generated token for the container of the operation.

[1] Note that, a formal definition of the mapping can be based on the MOF/XMI for data exchange of models; however for the sake of readability we use the graphical notation of the elements instead.

Lost and *found* messages are specified as usual messages with the exception of applying the wait-construct. *Co-regions* are constructed with the `flow`-construct and the messages of the co-region are transformed in the usual way.
One of the newest aspect of UML 2 sequence diagrams is the possibility to define combined fragements, depicted as 

. UML 2 distinguishes between:

- `alt` – at most one of the operands will execute, this can be transformed using the `switch`-construct




```
transform (          ) =
 <switch> <case condition="bool-expr">  transform(operand_1) </case>
                  …
           <otherwise>? transform(operand_n+1) </otherwise>
 </switch>
```

  in this case the alternatives in the sequence diagram have to be annotated with specific conditions for each case (as in our application example in one case).
- `opt` – either the (sole) operand happens or nothing happens, this is modeled similar to the `alt`-operator, where we have two cases, one case is the transformed operand and the other one is the distinguished no-operation of BPEL4WS.
- `loop` – repeated a number of times, transformed using the while-construct

```
<while condition="myConstraint"> transform(operand_loop) </while>
```

  where `myConstraint` is the translated constraint of the sequence diagram.
- `par` – parallel merge between the behaviors of the operands, this can easily be transformed with the flow-construct.

```
<flow> transform(operand_1) … transform(operand_n) </flow>
```

- `seq` – weak sequencing depending on lifelines and operands and strict – strict sequencing not depending on lifelines and operands can be modeled with

```
<sequence> transform(operand_1) … transform(operand_n) </sequence>
```

`critical` – critical region, this is handled by the transaction mechanism; `assert` – assertions are translated into boolean expressions which are evaluated during runtime; `ignore` – message types are not shown within `fragment`; `consider` – messages considered within fragment; and interaction reference – a `reference` to another interaction, can be seen as abbreviations and need not be transformed; `neg` – invalid traces, have not been transformed. Another novelty is the usage of continuations which can be seen as conditional "goto" statements. These continuations can be mapped to BPEL4WS by applying while-loops and a boolean global variable stating if the jump has to performed or not. The while-statement has to be placed at the maximal comprehensive block of the operands where the jump is performed.

4 Conclusions and Outlook

The main contribution of this paper is that it elaborates the relationship between the platform independent model of service choreography and its mapping to BPEL4WS a

specific business process execution language. The work is part of a larger project depicted in Figure 1. The informal definition of a mapping between the two representation shows that such a step can be automated. However additional information concerning the Web Services has to be at hand. This can be the WSDL definition of the Web Service interfaces specified with UML class diagrams as used e.g. by [20]. The next steps are the definition of a formal mapping between both representations; looking at a inverse mapping to allow reverse engineering; taking the "other" aspects of BPEL4WS into consideration, i.e. modelling of the context of the Web Service choreography; integration with the mapping from the computational independent model to the platform specific model, and integration into a development tool.

References

1. IBM (2003) 'Web Service Tutorial', http://www-106.ibm.com/developerworks/web/library/w-ovr/?dwzone=ibm
2. WebServices (2003) http://www.webservices.org
3. Sun (2002) 'Powering the Collaborative Enterprise Sun ONE and Java Technology in the Extended Supply Chain', http://www.sun.com/products-n-solutions/automotive/docs/sunarc.pdf
4. IBM (2003) 'BPEL4WS', http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
5. IBM (2003)' WSCI', http://wwws.sun.com/software/xml/developers/wsci/
6. WFMC (2003) 'XPDL', 10 June, http://www.wfmc.org/standards/docs.htm
7. ebXML (2003) 'Business Process Specification Schema', 10 June, http://www.ebxml.org/specs/ebBPSS.pdf
8. ebXML (2003) 'Enabling global electronic markets', http://www.ebxml.org
9. BPMI (2003) 'BPML', 10 June, http://www.bpmi.org/
10. MDA homepage. The Object Management Group (OMG). http://www.omg.org/
11. Kleppe M., Warmer J., Bast W. MDA Explained – The Model Driven Architecture: Practice and Promise, Addison Wesley, 2003
12. UML Homepage. The Object Management Group. http://www.omg.org/uml/
13. WFMC (2003) 'XPDL', 10 June, http://www.wfmc.org/standards/docs.htm
14. FIPA (2003), FIPA specifications, 10 June, http://www.fipa.org/specs/fipa00030/
15. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems, International Journal on Software Engineering and Knowledge Engineering (IJSEKE), Vol. 11, No. 3, 2001 Engineering, 2001.
16. Fuchs, I. (2002) 'Web Services and Business Process Management Platforms – Understanding Their Relationship and Defining an Implementation Approach', http://www.ebpml.org/ihf.doc
17. W3C Web Services glossary. http://www.w3.org/TR/ws-gloss/
18. Müller, J.P., Bauer, B., Friese, Th.: Programming software agents as designing executable business processes: a model-driven perspective, in Proceeding PROMAS 03, 2004.
19. Bauer, B., Marc-Philippe Huget: Modelling Web Service Composition with (Agent) UML, Special Issue of Journal of Web Engineering, 2003
20. Armstrong, Ch. (2002) 'Modelling Web Services with UML', Talk given at the OMG Web Services Workshop 2002.